**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 2

**З дисципліни:** *“Моделювання та аналіз програмного забезпечення”*

**На тему:** *“*Класи, інтерфейси і структуриу мові програмування C# *”*

**Лектор:**

доц. каф. ПЗ

Сердюк П.В.

**Виконав:**

ст. гр. ПЗ-23

Михалевич П.-І.В.

**Прийняла:**

асист. каф. ПЗ

Цимбалюк Т.М.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2022 р.

∑= \_\_\_\_\_ .

Львів – 2022

**Тема роботи:** Класи, інтерфейси і структури у мові програмування C#.

**Мета роботи:** Ознайомлення з основами класів, структур, та інших базових елементів мови програмування С#.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

**Клас**

Клас - основний тип даних в мові C#. Клас являє собою конструкцію, яка об’єднує поля, властивості та методи. Клас є визначенням для створення об’єктів або екземплярів класу.

Клас представляє собою шаблон, за яким визначається форма об’єкту. Клас є подібним до типів в традиційних мовах програмування. Відмінність полягає в тому, що користувач може створювати власні типи (класи) в об’єктно-орієнтованому програмуванні.

Якщо клас є подібним до типу в традиційних мовах програмування, тоді об’єкт є подібним до змінної. Інколи об’єкти називаються екземплярами класу. Об’єкт класу реалізує один з можливих варіантів опису класу. При оголошенні об’єкту класу для нього виділяється пам’ять.

**Інтерфейс**

Інтерфейс визначає ряд методів (властивостей, індексаторів, подій), які повинні бути реалізовані в класі що успадковує (реалізує) даний інтерфейс. Інтерфейси використовуються для того, щоб вказати класам, що саме повинно бути реалізоване в цих класах. Реалізовувати потрібно методи (властивості, індексатори, події). Таким чином, інтерфейс описує функціональні можливості без конкретної реалізації. Іншими словами, інтерфейс визначає специфікацію але не реалізацію.

Поняття інтерфейсу очевидне на звичайному життєвому рівні. Уявіть собі наприклад, плеєр для програвання компакт-дисків. Існує незліченна кількість різних моделей таких плеєрів, що відрізняються один від одного формою корпусу, кольором, розміром і іншими атрибутами. Проте всі вони мають практично однаковий набір кнопок, за допомогою яких можна запускати або зупиняти програвання компакт-диска, переходити з однієї доріжки на іншу, а також витягувати компакт-диск з корпусу плеєра.

Використовуючи інтерфейси, ви можете, наприклад, включити в клас поведінку з кількох джерел. Ця можливість важлива в C#, оскільки мова не підтримує багаторазове успадкування класів. Крім того, ви повинні використовувати інтерфейс, якщо ви хочете імітувати спадкування для структур, оскільки вони не можуть успадковувати від іншої структури або класу.

**ЗАВДАННЯ**

Перша частина

* Реалізувати ланцюжок наслідування у якому б був звичайний клас, абстрактний клас та інтерфейс. Перелічіть відмінності та подібності у цих структурах у звіті у вигляді таблиці.
* Реалізувати різні модифікатори доступу. Продемонструвати доступ до цих модифікаторів там де він є, та їх відсутність там, де це заборонено (включити в звіт вирізки з скріншотів Intelisense з VisualStudio).
* Реалізувати поля та класи без модифікаторів доступу. Дослідити який буде доступ за замовчуванням у класів, структур інтерфейсів, вкладених класів, полів, і т.д. У звіті має бути відповідна таблиця.
* Оголосити внутрішній клас з доступом меншим за public. Реалізувати поле цього типу даних. Дослідити обмеження на модифікатор.
* Реалізувати перелічуваний тип. Продемонструвати різні булівські операції на перелічуваних типах(^,||, &amp;&amp;. &amp;, |,…).
* Реалізувати множинне наслідування у C#.
* Реалізувати перевантаження конструкторів базового класу та поточного класу. Показати різні варіанти використання ключових слів base та this. Реалізувати перевантаження функції.
* Реалізувати різні види ініціалізації полів як статичних так і динамічних: за допомогою статичного та динамічного конструктора, та ін. Дослідити у якій послідовності ініціалізуються поля.
* Реалізувати функції з параметрами out, ref. Показати відмінності при наявності та без цих параметрів. Показати випадки, коли ці параметри не мають значення.
* Продемонуструвати boxing / unboxing.
* Реалізувати явні та неявні оператори приведення то іншого типу (implicit та explicit).
* Реалізувати логіку свого завдання у системі класів.
* Скопіювати проект і перейменувати всі класи у структури. Дослідити відмінності у класах та структурах та записати їх у вигляді таблиці до звіту. Реалізувати наслідування структур через інтерфейси.
* Перевизначити і дослідити методи класу object (у тому числі і protected методи).

Друга частина

Дослідити швидкодію структур у відповідності з завданням.

Розглянемо, для прикладу, завдання у порівнянні структур і класів. Для порівняльного аналізу створюємо клас і структуру з ідентичними полями. У програмі створюємо велику кількість об’єктів одного та іншого типу та вимірюємо зміни оперативної пам’яті, часу ініціалізації, тощо.

За замовчуванням для експериментів брати 10 млн однакових операцій, але якщо виникають проблеми з швидкодією чи визначенням часу, то орієнтуватись до можливостей власного комп’ютера .

Потрібно провести кілька обчислень у різний час (оскільки інші процеси можуть вплинути на результат) і узяти середнє значення відповідних вихідних даних.

Для вимірювання часу рекомендується використати клас Stopwatch.

**Індивідуальне завдання**

Варіант 10. Порівняти швидкість операцій boxing/unboxing та звичайного присвоєння для різних типів (int, double, float). Дослідити залежність часу операцій boxing/unboxing для структур із різною кількістю полів.

**ХІД ВИКОНАННЯ**

1. Реалізація ланцюжка наслідування із звичайного класу, абстрактного клас та інтерфейсу.

abstract class MyAbstractClass

{

public abstract void AbstractMethod();

}

interface MyInterface

{

void InterfaceMethod();

}

class MyClass: MyAbstractClass, MyInterface

{

void MyInterface.InterfaceMethod()

{

throw new NotImplementedException();

}

public override void AbstractMethod()

{

throw new NotImplementedException();

}

}

Табл. 1.

Порівняння абстрактного класу та інтерфейсу

|  |  |
| --- | --- |
| Абстрактний клас | Інтерфейс |
| Може містить як тільки оголошення, так і реалізацію деяких методів | Містить тільки оголошення (в нових версіях С# може бути реалізація) |
| Можна наслідувати від абстрактного класу та інтерфейсу | Можна наслідувати лише від інтерфейсу |
| Може мати конструктор | Не має конструктора |
| Може використовувати різні модифікатори доступу (public, private, protected) | Використовується тільки public |
| Клас наслідник не повинен реалізувувати всі методи, тільки ті, що потрібні | Клас наслідник повинен реалізовувати всі методи |
| Заборонено множинне наслідування | Дозволено множинне наслідування |

Табл. 2.

Порівняння класу та абстрактного класу

|  |  |
| --- | --- |
| Клас | Абстрактний клас |
| Не може містити абстрактних методів | Може містити абстрактні методи |
| Можна створити об'єкт | Не можна створити об’єкт |
| При наслідуванні від інтерфейсу повинен реалізувати всі методи інтерфейсу | При наслідуванні від інтерфейсу не повинен реалізовувати всі методи інтерфейсу |

2. Реалізація різних модифікаторів доступу.

class BaseClass

{

private int privateInt;

protected int protectedInt;

public int publicInt;

public BaseClass()

{

}

}

class FirstClass

{

public FirstClass()

{

BaseClass bc = new BaseClass();

bc.publicInt = 5;

bc.protectedInt = 5;

bc.privateInt = 5;

}

}

class SecondClass : BaseClass

{

public SecondClass()

{

publicInt = 5;

protectedInt = 5;

privateInt = 5;

}

}

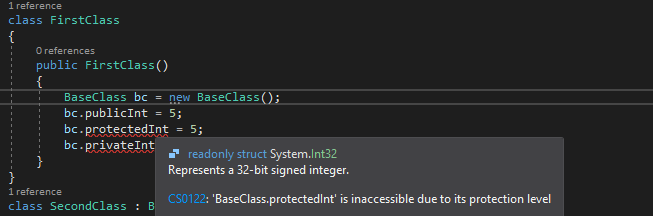


Рис.1. Помилка при звертанні до недоступних даних

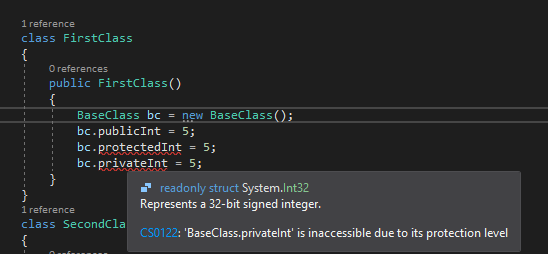


Рис.2. Помилка при звертанні до недоступних даних

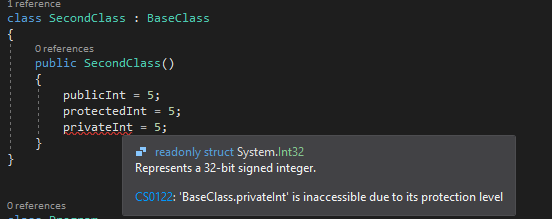


Рис.3. Помилка при звертанні до недоступних даних

3. Реалізація полів та класів без модифікаторів доступу.

Табл. 3.

Таблиця модифікаторів доступу по замовчуванню

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Struct | Методи | Вкладені об’єкти |
| public | public | public | private |

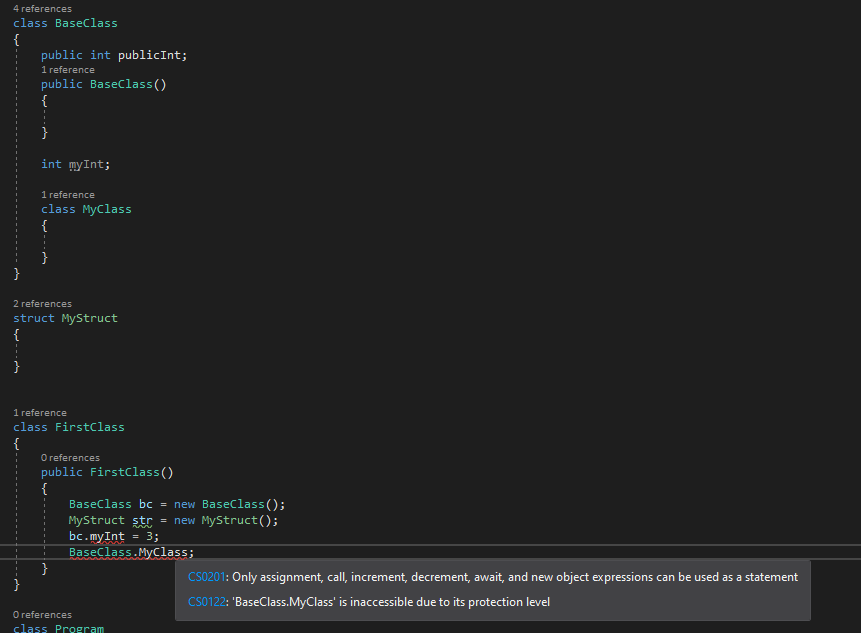


Рис.4. Помилка при звертанні до недоступних даних

4. Оголошення внутрішнього класу з доступом меншим за public. Реалізація поля цього типу даних.

class BaseClass

{

public int publicInt;

public BaseClass()

{

}

int myInt;

class MyClass

{

}

}

struct MyStruct

{

}

class FirstClass

{

public FirstClass()

{

BaseClass bc = new BaseClass();

MyStruct str = new MyStruct();

bc.myInt = 3;

BaseClass.MyClass;

}

}

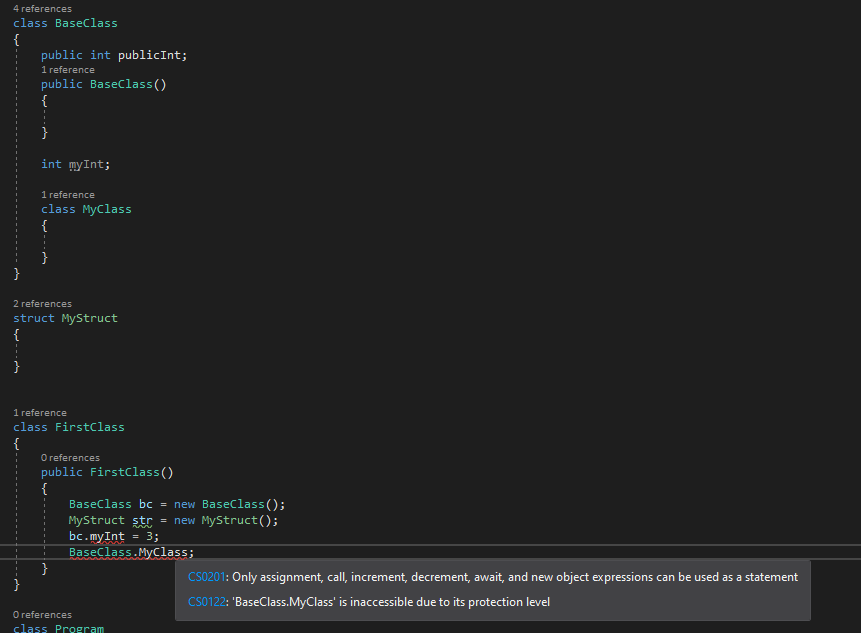


Рис.5. Помилка при звертанні до недоступних даних

5. Реалізація enum.

enum Food

{

Apple,

Banana,

Orange,

Potato,

Cherry,

Tomato,

Cucumber,

Fruits = Apple | Banana | Orange | Cherry,

Vegetables = Potato | Tomato | Cucumber

}

class Program

{

static void Main(string[] args)

{

bool isCherryFruit = (Food.Fruits & Food.Cherry) == Food.Cherry;

if (isCherryFruit)

{

Console.WriteLine("Cherry is fruit");

}

bool isPotatoVEgetable = (Food.Vegetables & Food.Potato) == Food.Potato;

if (isPotatoVEgetable)

{

Console.WriteLine("Potato is vegetable");

}

}

}

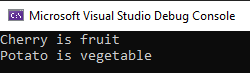


Рис.6. Виконання програми

6. Реалізація множинного наслідування.

abstract class MyAbstractClass

{

public abstract void AbstractMethod();

}

interface MyInterface

{

void InterfaceMethod();

}

interface MyInterface2

{

void SecondInterfaceMethod();

}

class MyClass : MyAbstractClass, MyInterface, MyInterface2

{

void MyInterface.InterfaceMethod()

{

throw new NotImplementedException();

}

void MyInterface2.SecondInterfaceMethod()

{

throw new NotImplementedException();

}

public override void AbstractMethod()

{

throw new NotImplementedException();

}

}

7. Реалізація перевантаження конструкторів базового класу та поточного класу.

class BaseClass

{

public int publicInt;

public BaseClass()

{

publicInt = 3;

}

}

class Class : BaseClass

{

public int myInt;

public Class() : base()

{

myInt = 4;

}

public Class(int n) : this()

{

myInt = n;

}

static void Main(string[] args)

{

Class myClass = new Class();

Console.WriteLine("Base class int = {0}\nDerived class int = {1}", myClass.publicInt, myClass.myInt);

}

}

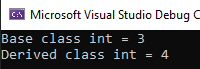


Рис.7. Виконання програми

8. Реалізація різних видів ініціалізації полів, як статичних так і динамічних: за допомогою статичного та динамічного конструктора.

class MyClass

{

// Static variable

static int s;

// Non-static variable

int ns;

// Declaration of

// static constructor

static MyClass()

{

s = 1;

Console.WriteLine("It is static constructor");

}

// Declaration of

// non-static constructor

public MyClass()

{

ns = 2;

Console.WriteLine("It is non-static constructor");

}

static void Main(string[] args)

{

MyClass obj1 = new MyClass();

MyClass obj2 = new MyClass();

}

}

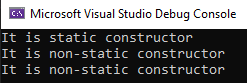


Рис.8. Виконання програми

9. Реалізація функцій з параметрами out, ref.

class MyClass

{

public void set4(int value)

{

value = 4;

}

public void set5(ref int value)

{

value = 5;

}

public void set6(out int value)

{

value = 6;

}

}

class Program

{

static void Main(string[] args)

{

int a = 1, b = 2, c;

MyClass class = new MyClass();

class.set4(a);

class.set5(ref b);

class.set6(out c);

Console.WriteLine($"{a} {b} {c}");

}

}



Рис.9. Вивід виконання програми

10. Демонстрування boxing / unboxing.

class Program

{

static void Main(string[] args)

{

int i = 123; // a value type

object o = i; // boxing

int j = (int)o; // unboxing

Console.WriteLine($"i = {i} \no = {o} \nj = {j}");

}

}

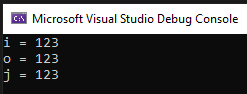


Рис.10. Вивід виконання програми

11. Реалізація явних та неявних операторів приведення до іншого типу.

Використання implicit:

public class Role

{

public string Name { get; set; }

public static implicit operator Role(string roleName)

{

return new Role() { Name = roleName };

}

static void Main(string[] args)

{

Role role = "RoleName";

Console.WriteLine($"Name = {role.Name}");

}

}

Використання explicit:

public class Role

{

public string Name { get; set; }

public static explicit operator Role(string roleName)

{

return new Role() { Name = roleName };

}

static void Main(string[] args)

{

Role role = (Role)"RoleName";

Console.WriteLine($"Name = {role.Name}");

}

}



Рис.11. Вивід виконання програми

12. Реалізація структури класів відповідно до індивідуального завдання.

Варіант 10. Порівняти швидкість операцій boxing/unboxing та звичайного присвоєння для різних типів (int, double, float). Дослідити залежність часу операцій boxing/unboxing для структур із різною кількістю полів.

class Class4

{

public int a1;

public int a2;

public int a3;

public int a4;

public Class4()

{

a1 = 1;

a2 = 2;

a3 = 3;

a4 = 4;

}

}

class Class3

{

public int b1;

public int b2;

public int b3;

public Class3()

{

b1 = 5;

b2 = 6;

b3 = 7;

}

}

class Class2

{

public int c1;

public int c2;

public Class2()

{

c1 = 8;

c2 = 9;

}

}

class Class1

{

public int d;

public Class1()

{

d = 0;

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Using one thousand operations for classes.");

Stopwatch sw = new Stopwatch();

sw.Start();

Class1 a = new Class1();

for (int m = 1; m < 1000; ++m)

{

Class1 a1 = new Class1();

}

sw.Stop();

Console.WriteLine("Class1 assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_a = (object)a;

for (int m = 1; m < 1000; ++m)

{

object \_\_a = (object)a;

}

sw.Stop();

Console.WriteLine("Class1 boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000; ++m)

{

Class1 a2 = (Class1)\_a;

}

sw.Stop();

Console.WriteLine("Class1 unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

//Class2

sw.Restart();

Class2 b = new Class2();

for (int m = 1; m < 1000; ++m)

{

Class2 b1 = new Class2();

}

sw.Stop();

Console.WriteLine("Class2 assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_b = (object)b;

for (int m = 1; m < 1000; ++m)

{

object \_\_b = (object)b;

}

sw.Stop();

Console.WriteLine("Class2 boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000; ++m)

{

Class2 b2 = (Class2)\_b;

}

sw.Stop();

Console.WriteLine("Class2 unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

//Class3

sw.Restart();

Class3 c = new Class3();

for (int m = 1; m < 1000; ++m)

{

Class3 c1 = new Class3();

}

sw.Stop();

Console.WriteLine("Class3 assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_c = (object)c;

for (int m = 1; m < 1000; ++m)

{

object \_\_c = (object)c;

}

sw.Stop();

Console.WriteLine("Class3 boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000; ++m)

{

Class3 c2 = (Class3)\_c;

}

sw.Stop();

Console.WriteLine("Class3 unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

//Class4

sw.Restart();

Class1 d = new Class1();

for (int m = 1; m < 1000; ++m)

{

Class4 d1 = new Class4();

}

sw.Stop();

Console.WriteLine("Class4 assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_d = (object)d;

for (int m = 1; m < 1000; ++m)

{

object \_\_d = (object)d;

}

sw.Stop();

Console.WriteLine("Class4 boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000; ++m)

{

Class4 d2 = (Class4)\_d;

}

sw.Stop();

Console.WriteLine("Class4 unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

//Types

Console.WriteLine("Using one bilion operations for types.");

sw.Restart();

sw.Start();

for(int m = 0; m < 1000000000; ++m)

{

int myint = 123123123;

}

sw.Stop();

Console.WriteLine("Int assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_myint = (object)123123123;

for (int m = 1; m < 1000000000; ++m)

{

object \_\_myint = (object)123123123;

}

sw.Stop();

Console.WriteLine("Int boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000000000; ++m)

{

int myint\_ = (int)\_myint;

}

sw.Stop();

Console.WriteLine("Int unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000000000; ++m)

{

double i = 123123123.123123123123;

}

sw.Stop();

Console.WriteLine("Double assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object o = (object)123123123.123123123123;

for (int m = 1; m < 1000000000; ++m)

{

object \_o = (object)123123123.123123123123;

}

sw.Stop();

Console.WriteLine("Double boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000000000; ++m)

{

double j = (double)o;

}

sw.Stop();

Console.WriteLine("Double unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000000000; ++m)

{

float myfloat = 123123123.123123F;

}

sw.Stop();

Console.WriteLine("Float assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_myfloat = (object)123123123.123123F;

for (int m = 1; m < 1000000000; ++m)

{

object \_\_myfloat = (object)123123123.123123F;

}

sw.Stop();

Console.WriteLine("Float boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000000000; ++m)

{

float myfloat\_ = (float)\_myfloat;

}

sw.Stop();

Console.WriteLine("Float unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

}

}

13. Перейменування класів у структури.

struct Class4

{

public int a1;

public int a2;

public int a3;

public int a4;

}

struct Class3

{

public int b1;

public int b2;

public int b3;

}

struct Class2

{

public int c1;

public int c2;

}

struct Class1

{

public int d;

}

Табл. 4.

Порівняння класів та структур

|  |  |
| --- | --- |
| Клас | Структура |
| Сукупність даних і функцій | |
| Тип за посиланням(купа) | Тип за значенням(стек) |
| Можуть наслідувати класи і інтерфейси | Можуть наслідувати тільки інтерфейси |
| Можна превизначити конструктор за замовчуванням | Не можна превизначити конструктор за замовчуванням |
| Ініціалізовувати поля дозволено | Ініціалізовувати поля заборонено |

14. Перевизначення функцій object.

class Point

{

public int x, y;

public Point(int x, int y)

{

this.x = x;

this.y = y;

}

public override bool Equals(object obj)

{

if (obj.GetType() != this.GetType()) return false;

var other = (Point)obj;

return (this.x == other.x) && (this.y == other.y);

}

public override int GetHashCode()

{

return x ^ y;

}

public override String ToString()

{

return $"({x}, {y})";

}

static void Main()

{

var p1 = new Point(1, 2);

var p3 = p1;

Console.WriteLine(Object.Equals(p1, p3));

Console.WriteLine($"p1's value is: {p1.ToString()}");

}

}

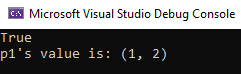


Рис.12. Вивід виконання програми

Частина друга

Варіант 10. Порівняти швидкість операцій boxing/unboxing та звичайного присвоєння для різних типів (int, double, float). Дослідити залежність часу операцій boxing/unboxing для структур із різною кількістю полів.

Код для реалізації даної задачі:

struct Class4

{

public int a1;

public int a2;

public int a3;

public int a4;

}

struct Class3

{

public int b1;

public int b2;

public int b3;

}

struct Class2

{

public int c1;

public int c2;

}

struct Class1

{

public int d;

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Using ten milions operations for classes.");

Stopwatch sw = new Stopwatch();

sw.Start();

Class1 a = new Class1();

for (int m = 1; m < 10000000; ++m)

{

Class1 a1 = new Class1();

}

sw.Stop();

Console.WriteLine("Class1 assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_a = (object)a;

for (int m = 1; m < 10000000; ++m)

{

object \_\_a = (object)a;

}

sw.Stop();

Console.WriteLine("Class1 boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 10000000; ++m)

{

Class1 a2 = (Class1)\_a;

}

sw.Stop();

Console.WriteLine("Class1 unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

//Class2

sw.Restart();

Class2 b = new Class2();

for (int m = 1; m < 10000000; ++m)

{

Class2 b1 = new Class2();

}

sw.Stop();

Console.WriteLine("Class2 assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_b = (object)b;

for (int m = 1; m < 10000000; ++m)

{

object \_\_b = (object)b;

}

sw.Stop();

Console.WriteLine("Class2 boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 10000000; ++m)

{

Class2 b2 = (Class2)\_b;

}

sw.Stop();

Console.WriteLine("Class2 unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

//Class3

sw.Restart();

Class3 c = new Class3();

for (int m = 1; m < 10000000; ++m)

{

Class3 c1 = new Class3();

}

sw.Stop();

Console.WriteLine("Class3 assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_c = (object)c;

for (int m = 1; m < 10000000; ++m)

{

object \_\_c = (object)c;

}

sw.Stop();

Console.WriteLine("Class3 boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 10000000; ++m)

{

Class3 c2 = (Class3)\_c;

}

sw.Stop();

Console.WriteLine("Class3 unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

//Class4

sw.Restart();

Class4 d = new Class4();

for (int m = 1; m < 10000000; ++m)

{

Class4 d1 = new Class4();

}

sw.Stop();

Console.WriteLine("Class4 assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_d = (object)d;

for (int m = 1; m < 10000000; ++m)

{

object \_\_d = (object)d;

}

sw.Stop();

Console.WriteLine("Class4 boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 10000000; ++m)

{

Class4 d2 = (Class4)\_d;

}

sw.Stop();

Console.WriteLine("Class4 unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

//Types

Console.WriteLine("Using one bilion operations for types.");

sw.Restart();

sw.Start();

for (int m = 0; m < 1000000000; ++m)

{

int myint = 123123123;

}

sw.Stop();

Console.WriteLine("Int assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_myint = (object)123123123;

for (int m = 1; m < 1000000000; ++m)

{

object \_\_myint = (object)123123123;

}

sw.Stop();

Console.WriteLine("Int boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000000000; ++m)

{

int myint\_ = (int)\_myint;

}

sw.Stop();

Console.WriteLine("Int unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000000000; ++m)

{

double i = 123123123.123123123123;

}

sw.Stop();

Console.WriteLine("Double assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object o = (object)123123123.123123123123;

for (int m = 1; m < 1000000000; ++m)

{

object \_o = (object)123123123.123123123123;

}

sw.Stop();

Console.WriteLine("Double boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000000000; ++m)

{

double j = (double)o;

}

sw.Stop();

Console.WriteLine("Double unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000000000; ++m)

{

float myfloat = 123123123.123123F;

}

sw.Stop();

Console.WriteLine("Float assign: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

object \_myfloat = (object)123123123.123123F;

for (int m = 1; m < 1000000000; ++m)

{

object \_\_myfloat = (object)123123123.123123F;

}

sw.Stop();

Console.WriteLine("Float boxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

sw.Restart();

for (int m = 0; m < 1000000000; ++m)

{

float myfloat\_ = (float)\_myfloat;

}

sw.Stop();

Console.WriteLine("Float unboxing: " + sw.ElapsedMilliseconds.ToString() + "miliseconds");

}

}

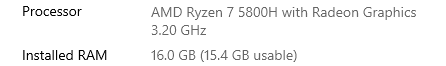


Рис.13. Характеристика комп'ютера

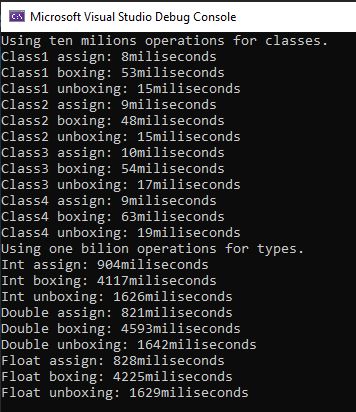


Рис.14. Вивід виконання програми

Табл. 5.

Таблиця з випробуваннями для типів (1 мільярд операцій)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Тип операції | №1, мс | №2, мс | №3, мс | №4, мс | №5, мс |
| Int assign | 904 | 911 | 830 | 826 | 829 |
| Int boxing | 4117 | 4690 | 4407 | 4239 | 4135 |
| Int unboxing | 1626 | 1814 | 1708 | 1676 | 1684 |
| Float assign | 821 | 837 | 873 | 866 | 855 |
| Float boxing | 4593 | 4750 | 4428 | 4461 | 4377 |
| Float unboxing | 1642 | 1764 | 1657 | 1686 | 1692 |
| Double assign | 828 | 848 | 832 | 852 | 860 |
| Double boxing | 4225 | 4991 | 4318 | 4404 | 4507 |
| Double unboxing | 1629 | 1784 | 1647 | 1697 | 1679 |

Табл. 6.

Таблиця з середніми значеннями випробувань для типів (1 мільярд операцій)

|  |  |
| --- | --- |
| Тип операції | Середнє значення, мс |
| Int assign | 860.00 |
| Int boxing | 4317.6 |
| Int unboxing | 1701.6 |
| Float assign | 850.4 |
| Float boxing | 4521.8 |
| Float unboxing | 1688.2 |
| Double assign | 844 |
| Double boxing | 4489 |
| Double unboxing | 1687.2 |

Class 1 – структура з 1 полем;

Class 2 – структура з 2 полями;

Class 3 – структура з 3 полями;

Class 4 – структура з 4 полями;

Табл. 7.

Таблиця з випробуваннями для структур (10 млн операцій)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Тип операції | №1, мс | №2, мс | №3, мс | №4, мс | №5, мс |
| Class1 assign | 8 | 8 | 8 | 8 | 8 |
| Class1 boxing | 53 | 51 | 51 | 47 | 47 |
| Class1 unboxing | 15 | 14 | 14 | 14 | 14 |
| Class2 assign | 9 | 8 | 8 | 8 | 8 |
| Class2 boxing | 48 | 49 | 53 | 46 | 49 |
| Class2 unboxing | 15 | 15 | 15 | 14 | 14 |
| Class3 assign | 10 | 10 | 9 | 10 | 9 |
| Class3 boxing | 54 | 52 | 50 | 52 | 54 |
| Class3 unboxing | 17 | 17 | 17 | 17 | 17 |
| Class4 assign | 9 | 9 | 8 | 9 | 8 |
| Class4 boxing | 63 | 62 | 61 | 58 | 77 |
| Class4 unboxing | 19 | 18 | 17 | 17 | 17 |

Табл. 8.

Таблиця з середніми значеннями випробувань для структур (10 млн операцій)

|  |  |
| --- | --- |
| Тип операції | Середнє значення, мс |
| Class1 assign | 8 |
| Class1 boxing | 49.8 |
| Class1 unboxing | 14.2 |
| Class2 assign | 8.2 |
| Class2 boxing | 49 |
| Class2 unboxing | 14.6 |
| Class3 assign | 9.6 |
| Class3 boxing | 52.4 |
| Class3 unboxing | 17 |
| Class4 assign | 8.6 |
| Class4 boxing | 64.2 |
| Class4 unboxing | 17.6 |

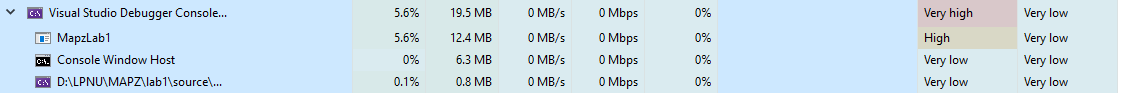


Рис.15. Використання оперативної пам’яті

З отриманих результатів можна зробити висновок, що для всіх звичайних типів або типів створених користувачем операція присвоєння завжди швидша приблизно 2 рази від операції unboxing. При реалізації boxing/unboxing найбільше часу витрачається саме на boxing – приблизно ¾ від часу усієї операції.

Для системних типів (int, double, float) операції boxing/unboxing та звичайного прискорення найшвидші у int (якщо не враховувати похибку у перших двої експериментах), оскільки він займає найменшу кількість пам’яті і не має дробової частини. Далі йде double, оскільки у C# цей тип більш оптимізований, аніж float. І останній тип це float.

Для користувацьких структур час кожної операції збільшується відповідно до розміру користувацького типу. Наприклад, оскільки в Class1 є тільки 1 поле, то усі операції з цією структурою будуть швидші, ніж операції зі структурою Class4, яка має 4 поля.

**ВИСНОВКИ**

На даній лабораторній роботі я ознайомився з класами, структурами, перелічуваними типами та інтерфейсами у мові C#, навчився як створювати власні типи даних та як оперувати над ним, ознайомився з усіма модифікаторами доступу і їхніми комбінаціями. Вивчив, що таке наслідування класів, boxing/unboxing, перевантаження методів, функції з out/ref параметрами і явні, неявні оператори приведення.

Дізнався, які є модифікатори доступу за замовчуванням, а саме: public у класів і методів, private у вкладених полів. Вивчив, що класи можуть наслідувати тільки один клас, але мають можливість множинного наслідування інтерфейсів, інтерфейси можуть наслідувати лише інтерфейси. Дізнався про оператори base і this за допомогою яких можна зручно використовувати перевантаження конструкторів класу. Навчився використовувати параметри out/ref, які дають змогу працювати з типами Value, як з силками, а також ознайомився з операторами implicit і explicit, що дають змогу робити явні та неявні приведеня відповідно.

Дізнався про різницю між інтерфейсами та абстрактними класами, одні з відмінностей:

1. У абстракному класі можна використовувати всі модифікатори доступи, а у інтрефейсах тільки public.
2. Клас наслідник абстрактного класу може не перевизначати всі методи, а клас наслідник інтерфейсу має перевантажити всі методи.

Також дізнався про різницю між структурами та класами:

1. Клас це тип за посиданням, коли структура це тип за значенням.
2. Клас може наслідувати класи і інтерфейси, а структури тільки інтерфейси.
3. В класі можна ініціалізувати поля, в структурі – ні.

У другій частині роботи ближче ознайомився з операцію boxing/unboxing і зрозумів, що вона менш ефективна, ніж звичайне присвоєння у звичайних умовах, однак наприклад у випадку, коли нам потрібно працювати з різними типами в одному тип масиві даних, object допомагає обійти потребу у створені нових масивів. Саме тут нам потрібна операція boxing/unboxing.